Segundo Trabalho Prático de Sistemas Operacionais

Arthur Antunes Santos Silva, Dilvonei Lacerda e Lucas Gonçalves Nojiri
Professor: Rafael Sachetto Oliveira
Departamento de Ciências da Computação – Universidade Federal de São João del-Rei - UFSJ – Campus Tancredo Neves

1- Introdução

Este trabalho teve como propósito a implementação de um simulador de um sistema de arquivos baseado na tabela FAT e um shell de comandos para realizar as operações sobre este sistema de arquivo.

2- Comandos shell

Comando	Descrição
init	inicializar o sistema de arquivos com as estruturas de dados, semelhante a formatar o sistema de arquivos virtual
load	carregar o sistema de arquivos do disco
ls	listar diretório
mkdir	criar diretório
create	criar arquivo
unlink	excluir arquivo ou diretório que esteja vazio
write	escrever dados em um arquivo
append	anexar dados em um arquivo
read	ler o conteúdo de um arquivo

3- Funções

3.1- void init()

Essa função inicializa o sistema de arquivos e define os valores do boot block em 0xbb, preenchendo a tabela FAT com os valores iniciais.

3.2- int load()

Nessa função é aberto o arquivo “fat.part” e preenchida a tabela FAT com os dados do arquivo.

3.3- void atualiza_fat()

Toda vez que essa função é chamada a tabela FAT é atualizada.

3.4- void save_cluster(int index, data_cluster* cluster)

Faz a gravação do cluster na memória.

3.5- data_cluster ler_cluster(int index) VOLTAR

Essa função abre o arquivo “fat.part” somente para leitura recebendo um inteiro corresponde ao que será lido e retorna este cluster.

3.6- data_cluster* procura_dir(data_cluster* cluster_atual, char* diretorio, int* addr)

Faz a procura do diretório em relação a string recebida e retorna o índice do cluster atual.

3.7- *int* num_dir(*char *caminho*);

Essa função recebe uma string que define o caminho de determinado diretório e percorre os diretórios do caminho.

3.8- *void* separa(*char *str1*, *char *str2*, *char *str3*, *char *separador*)

Esta função divide a string 1(str1) em duas partes: uma parte antes da primeira ocorrência do caractere separado e outra parte depois dessa ocorrência

Ela recebe três strings: a string 1 que será dividida, a segunda a string 2 que recebe a primeira parte da divisão e a terceira que é a string 3 que recebe a última parte da divisão. A função também recebe um caractere que determina a forma como a divisão irá ser realizada.

3.9- data_cluster *qtd_cluster(*char *string*, *int *numClusters*)

Esta função calcula a quantidade de clusters que serão necessários para armazenar uma string, cria um array de clusters que será preenchido com a string e retorna o array.

3.10- void mkdir(char* diretorio)

Essa função cria um novo diretório, seu parâmetro do tipo char irá definir o caminho desse novo diretório. É chamada a função procura_dir para busca o index do cluster do diretório pai, e então é feito a leitura do cluster que foi encontrado. Finalmente é verificado se existe um espaço que esteja livre para criação e a existência de um diretório de nome repetido, para então completar a criação tendo as condições cumpridas.

3.12- void ls(char* diretorio)

Essa função cria um novo diretório, seu parâmetro do tipo char irá definir o caminho desse novo diretório. É chamada a função

procura_dir para busca o index do cluster do diretório pai, e então é feito a leitura do cluster que foi encontrado. As 32 posições de entrada do cluster são percorridas, em seguida é impresso o nome do diretório ou uma mensagem de erro caso não seja encontrado.

3.13- void create(char* diretorio)

Nessa função é realizada a criação de um novo arquivo no caminho de determinado diretório. É chamada a função procura_dir para busca o index do cluster do diretório onde será salvo o arquivo, então é feito a leitura do cluster que foi encontrado. Finalmente é verificado se existe um espaço que esteja livre para criação e a existência de um arquivo de nome repetido, para então completar a criação tendo as condições cumpridas.

3.14- void write(char* diretorio, char content) VOLTAR

Esta função tem como parâmetro uma string, salvando o caminho do arquivo definido pela string , para encontrar o index do cluster, retornando os clusters para salvar a string, atualizando a tabela FAT.

3.15- void unlink(char* diretorio)

Nessa função é deletado o diretório ou arquivo desejado no caminho. É chamada a função procura_dir para busca o index do cluster do diretório pai onde está localizado o diretório ou o arquivo. É feito a leitura e verificado se o diretório ou o arquivo existem para que seja realizada a exclusão.

3.16- void read(char* diretorio)

Essa função fará a leitura de um arquivo, percorrendo os diretórios verificando a existência do arquivo. É exibido o endereço, caso exista, ou uma mensagem de erro, caso não exista.
Com isso o resto será separado nos clusters

3.17- void append(char* diretorio, char* content)

Essa função append recebe como parâmetro uma string que define o caminho do arquivo. É verificado se terá espaço para esta string criada, e se não for preenchido totalmente, será preenchido para o tamanho restante do cluster.

3.18- void hep()

Imprime os comandos do shell.

3.19- void command()

Essa função imprime o menu de comandos para o shell.

4- Testes

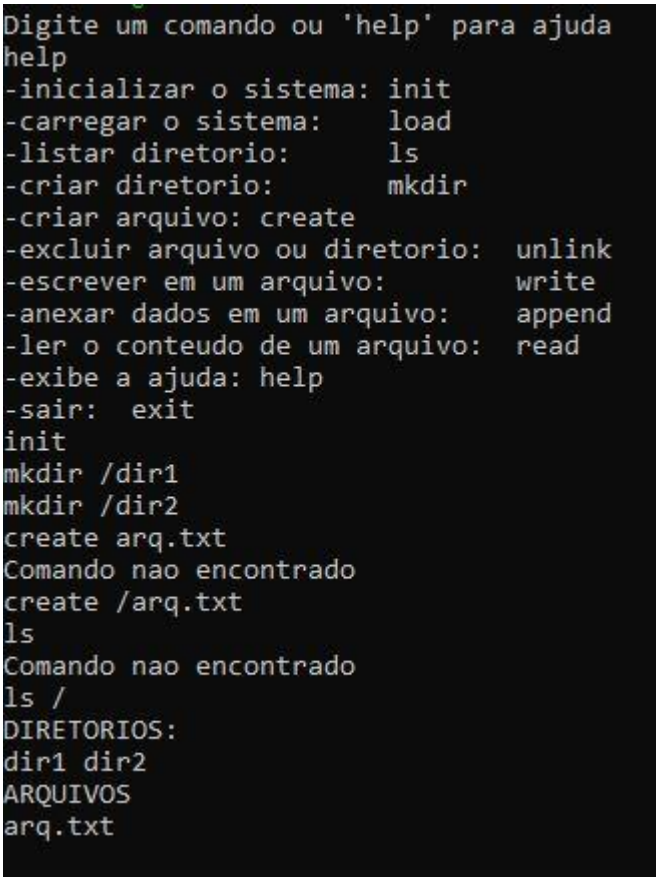

Figura 1 - Comandos: init, mkdir, create e ls para demonstrar que os arquivos foram criados corretamente.

```
init
mkdir /num1
mkdir /num2
create /arq.txt
load
ls /
num1
num2
arq.txt
write /arq.txt
Digite o texto TEST
read /arq.txt
 TEST

append /arq.txt
Digite o texto numero 03
read /arq.txt
 TEST
 numero 03
```

Figura 2 - Comandos:mkdir, load, write, read e append.

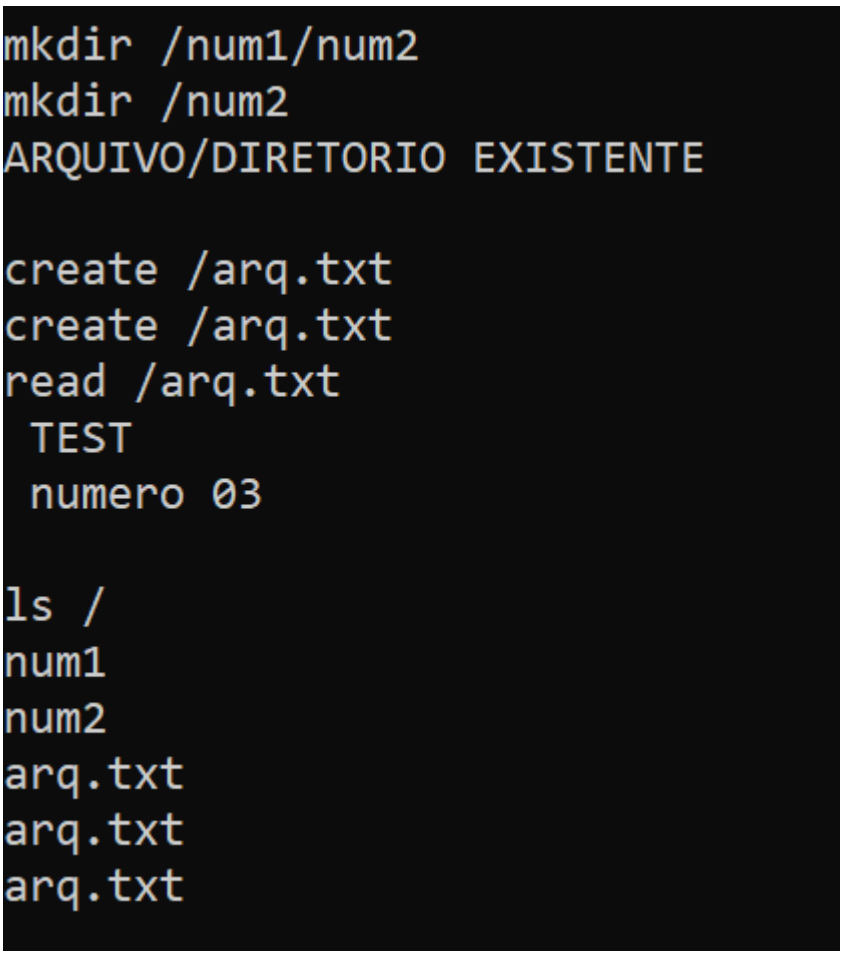

Figura 3 - Comandos:mkdir, create e read, com mensagens de erros, no entanto o create ainda criava arquivos .txt com nomes iguais.

5- Conclusão

Durante a realização deste trabalho foram encontradas várias dificuldades, entre elas a implementação de algumas funções, sendo a principal entre elas a procura_dir, sendo uma função muito importante para a execução dos comandos shell, outras funções como mkdir, append e unlink, também foram de grande dificuldade, mas que eventualmente com ajuda, as dúvidas foram respondidas.

Dessa forma foi possível a conclusão no tempo proposto e a fixação dos conteúdos do trabalho.

6- Referências

[1]Tanenbaum, Andrew S. Sistemas operacionais modernos / Andrew S. Tanenbaum, Herbert Bos
[2]Maziero, Carlos Alberto. Sistemas operacionais: conceitos e mecanismos / – Curitiba : DINF - UFPR, 2019

[3]Materiais postados no campusvirtual